NAME

jstat – monitor JVM statistics

SYNOPSIS

Note: This command is experimental and unsupported.

`jstat` *generalOptions*

`jstat` *outputOptions* [`-t`] [`-h` *lines*] *vmid* [*interval* [*count*]]

generalOptions
A single general command–line option. See **General Options**.

outputOptions
An option reported by the `-options` option. One or more output options that consist of a single `statOption`, plus any of the `-t`, `-h`, and `-J` options. See **Output Options for the jstat Command**.

`-t` Displays a time–stamp column as the first column of output. The time stamp is the time since the start time of the target JVM.

`-h` *n* Displays a column header every *n* samples (output rows), where *n* is a positive integer. The default value is `0`, which displays the column header of the first row of data.

vmid A virtual machine identifier, which is a string that indicates the target JVM. See **Virtual Machine Identifier**.

interval
The sampling interval in the specified units, seconds (s) or milliseconds (ms). Default units are milliseconds. This must be a positive integer. When specified, the `jstat` command produces its output at each interval.

count The number of samples to display. The default value is infinity, which causes the `jstat` command to display statistics until the target JVM terminates or the `jstat` command is terminated. This value must be a positive integer.

DESCRIPTION

The `jstat` command displays performance statistics for an instrumented Java HotSpot VM. The target JVM is identified by its virtual machine identifier, or `vmid` option.

The `jstat` command supports two types of options, general options and output options. General options cause the `jstat` command to display simple usage and version information. Output options determine the content and format of the statistical output.

All options and their functionality are subject to change or removal in future releases.

GENERAL OPTIONS

If you specify one of the general options, then you can't specify any other option or parameter.

`-help` Displays a help message.

`-options`
Displays a list of static options. See **Output Options for the jstat Command**.

OUTPUT OPTIONS FOR THE JSTAT COMMAND

If you don't specify a general option, then you can specify output options. Output options determine the content and format of the `jstat` command's output, and consist of a single `statOption`, plus any of the other output options (`-h`, `-t`, and `-J`). The `statOption` must come first.

Output is formatted as a table, with columns that are separated by spaces. A header row with titles describes the columns. Use the `-h` option to set the frequency at which the header is displayed. Column header names are consistent among the different options. In general, if two options provide a column with the same name, then the data source for the two columns is the same.

Use the `-t` option to display a time–stamp column, labeled Timestamp as the first column of output. The Timestamp column contains the elapsed time, in seconds, since the target JVM started. The resolution of

the time stamp is dependent on various factors and is subject to variation due to delayed thread scheduling on heavily loaded systems.

Use the interval and count parameters to determine how frequently and how many times, respectively, the `jstat` command displays its output.

Note:

Don't write scripts to parse the `jstat` command's output because the format might change in future releases. If you write scripts that parse the `jstat` command output, then expect to modify them for future releases of this tool.

`-statOption`
: Determines the statistics information that the `jstat` command displays. The following lists the available options. Use the `-options` general option to display the list of options for a particular platform installation. See **Stat Options and Output**.

  `class`: Displays statistics about the behavior of the class loader.

  `compiler`: Displays statistics about the behavior of the Java HotSpot VM Just−in−Time compiler.

  `gc`: Displays statistics about the behavior of the garbage collected heap.

  `gccapacity`: Displays statistics about the capacities of the generations and their corresponding spaces.

  `gccause`: Displays a summary about garbage collection statistics (same as `-gcutil`), with the cause of the last and current (when applicable) garbage collection events.

  `gcnew`: Displays statistics about the behavior of the new generation.

  `gcnewcapacity`: Displays statistics about the sizes of the new generations and their corresponding spaces.

  `gcold`: Displays statistics about the behavior of the old generation and metaspace statistics.

  `gcoldcapacity`: Displays statistics about the sizes of the old generation.

  `gcmetacapacity`: Displays statistics about the sizes of the metaspace.

  `gcutil`: Displays a summary about garbage collection statistics.

  `printcompilation`: Displays Java HotSpot VM compilation method statistics.

`-J`*javaOption*
: Passes *javaOption* to the Java application launcher. For example, `-J-Xms48m` sets the startup memory to 48 MB. For a complete list of options, see **java**.

STAT OPTIONS AND OUTPUT

The following information summarizes the columns that the `jstat` command outputs for each *statOption*.

`-class` *option*
: Class loader statistics.

  `Loaded`: Number of classes loaded.

  `Bytes`: Number of KB loaded.

  `Unloaded`: Number of classes unloaded.

  `Bytes`: Number of KB unloaded.

  `Time`: Time spent performing class loading and unloading operations.

`-compiler` *option*
: Java HotSpot VM Just−in−Time compiler statistics.

  `Compiled`: Number of compilation tasks performed.

  `Failed`: Number of compilations tasks failed.

`Invalid`: Number of compilation tasks that were invalidated.

`Time`: Time spent performing compilation tasks.

`FailedType`: Compile type of the last failed compilation.

`FailedMethod`: Class name and method of the last failed compilation.

`-gc` *option*

Garbage collected heap statistics.

`S0C`: Current survivor space 0 capacity (KB).

`S1C`: Current survivor space 1 capacity (KB).

`S0U`: Survivor space 0 utilization (KB).

`S1U`: Survivor space 1 utilization (KB).

`EC`: Current eden space capacity (KB).

`EU`: Eden space utilization (KB).

`OC`: Current old space capacity (KB).

`OU`: Old space utilization (KB).

`MC`: Metaspace Committed Size (KB).

`MU`: Metaspace utilization (KB).

`CCSC`: Compressed class committed size (KB).

`CCSU`: Compressed class space used (KB).

`YGC`: Number of young generation garbage collection (GC) events.

`YGCT`: Young generation garbage collection time.

`FGC`: Number of full GC events.

`FGCT`: Full garbage collection time.

`GCT`: Total garbage collection time.

`-gccapacity` *option*

Memory pool generation and space capacities.

`NGCMN`: Minimum new generation capacity (KB).

`NGCMX`: Maximum new generation capacity (KB).

`NGC`: Current new generation capacity (KB).

`S0C`: Current survivor space 0 capacity (KB).

`S1C`: Current survivor space 1 capacity (KB).

`EC`: Current eden space capacity (KB).

`OGCMN`: Minimum old generation capacity (KB).

`OGCMX`: Maximum old generation capacity (KB).

`OGC`: Current old generation capacity (KB).

`OC`: Current old space capacity (KB).

`MCMN`: Minimum metaspace capacity (KB).

`MCMX`: Maximum metaspace capacity (KB).

`MC`: Metaspace Committed Size (KB).

`CCSMN`: Compressed class space minimum capacity (KB).

`CCSMX`: Compressed class space maximum capacity (KB).

`CCSC`: Compressed class committed size (KB).

`YGC`: Number of young generation GC events.

`FGC`: Number of full GC events.

`-gccause` *option*
: This option displays the same summary of garbage collection statistics as the `-gcutil` option, but includes the causes of the last garbage collection event and (when applicable), the current garbage collection event. In addition to the columns listed for `-gcutil`, this option adds the following columns:

`LGCC`: Cause of last garbage collection

`GCC`: Cause of current garbage collection

`-gcnew` *option*
: New generation statistics.

`S0C`: Current survivor space 0 capacity (KB).

`S1C`: Current survivor space 1 capacity (KB).

`S0U`: Survivor space 0 utilization (KB).

`S1U`: Survivor space 1 utilization (KB).

`TT`: Tenuring threshold.

`MTT`: Maximum tenuring threshold.

`DSS`: Desired survivor size (KB).

`EC`: Current eden space capacity (KB).

`EU`: Eden space utilization (KB).

`YGC`: Number of young generation GC events.

`YGCT`: Young generation garbage collection time.

`-gcnewcapacity` *option*
: New generation space size statistics.

`NGCMN`: Minimum new generation capacity (KB).

`NGCMX`: Maximum new generation capacity (KB).

`NGC`: Current new generation capacity (KB).

`S0CMX`: Maximum survivor space 0 capacity (KB).

`S0C`: Current survivor space 0 capacity (KB).

`S1CMX`: Maximum survivor space 1 capacity (KB).

`S1C`: Current survivor space 1 capacity (KB).

`ECMX`: Maximum eden space capacity (KB).

`EC`: Current eden space capacity (KB).

`YGC`: Number of young generation GC events.

`FGC`: Number of full GC events.

`-gcold` *option*
: Old generation size statistics.

`MC`: Metaspace Committed Size (KB).

`MU`: Metaspace utilization (KB).

`CCSC`: Compressed class committed size (KB).

`CCSU`: Compressed class space used (KB).

`OC`: Current old space capacity (KB).

`OU`: Old space utilization (KB).

`YGC`: Number of young generation GC events.

`FGC`: Number of full GC events.

`FGCT`: Full garbage collection time.

`GCT`: Total garbage collection time.

`-gcoldcapacity` *option*
Old generation statistics.

`OGCMN`: Minimum old generation capacity (KB).

`OGCMX`: Maximum old generation capacity (KB).

`OGC`: Current old generation capacity (KB).

`OC`: Current old space capacity (KB).

`YGC`: Number of young generation GC events.

`FGC`: Number of full GC events.

`FGCT`: Full garbage collection time.

`GCT`: Total garbage collection time.

`-gcmetacapacity` *option*
Metaspace size statistics.

`MCMN`: Minimum metaspace capacity (KB).

`MCMX`: Maximum metaspace capacity (KB).

`MC`: Metaspace Committed Size (KB).

`CCSMN`: Compressed class space minimum capacity (KB).

`CCSMX`: Compressed class space maximum capacity (KB).

`YGC`: Number of young generation GC events.

`FGC`: Number of full GC events.

`FGCT`: Full garbage collection time.

`GCT`: Total garbage collection time.

`-gcutil` *option*
Summary of garbage collection statistics.

`S0`: Survivor space 0 utilization as a percentage of the space's current capacity.

`S1`: Survivor space 1 utilization as a percentage of the space's current capacity.

`E`: Eden space utilization as a percentage of the space's current capacity.

`O`: Old space utilization as a percentage of the space's current capacity.

`M`: Metaspace utilization as a percentage of the space's current capacity.

`CCS`: Compressed class space utilization as a percentage.

`YGC`: Number of young generation GC events.

`YGCT`: Young generation garbage collection time.

`FGC`: Number of full GC events.

`FGCT`: Full garbage collection time.

`GCT`: Total garbage collection time.

`-printcompilation` *option*
Java HotSpot VM compiler method statistics.

`Compiled`: Number of compilation tasks performed by the most recently compiled method.

`Size`: Number of bytes of byte code of the most recently compiled method.

`Type`: Compilation type of the most recently compiled method.

`Method`: Class name and method name identifying the most recently compiled method. Class name uses a slash (/) instead of a dot (.) as a name space separator. The method name is the method within the specified class. The format for these two fields is consistent with the HotSpot `-XX:+PrintCompilation` option.

VIRTUAL MACHINE IDENTIFIER

The syntax of the `vmid` string corresponds to the syntax of a URI:

[*protocol*`:`][`//`]*lvmid*[`@`*hostname*[`:`*port*][`/`*servername*]

The syntax of the `vmid` string corresponds to the syntax of a URI. The `vmid` string can vary from a simple integer that represents a local JVM to a more complex construction that specifies a communications protocol, port number, and other implementation−specific values.

protocol
The communications protocol. If the *protocol* value is omitted and a host name isn't specified, then the default protocol is a platform−specific optimized local protocol. If the *protocol* value is omitted and a host name is specified, then the default protocol is `rmi`.

lvmid The local virtual machine identifier for the target JVM. The *lvmid* is a platform−specific value that uniquely identifies a JVM on a system. The *lvmid* is the only required component of a virtual machine identifier. The *lvmid* is typically, but not necessarily, the operating system's process identifier for the target JVM process. You can use the `jps` command to determine the *lvmid* provided the JVM processes is not running in a separate docker instance. You can also determine the *lvmid* on Linux and macOS platforms with the `ps` command, and on Windows with the Windows Task Manager.

hostname
A host name or IP address that indicates the target host. If the *hostname* value is omitted, then the target host is the local host.

port The default port for communicating with the remote server. If the *hostname* value is omitted or the *protocol* value specifies an optimized, local protocol, then the *port* value is ignored. Otherwise, treatment of the *port* parameter is implementation−specific. For the default `rmi` protocol, the port value indicates the port number for the `rmiregistry` on the remote host. If the *port* value is omitted and the *protocol* value indicates `rmi`, then the default rmiregistry port (1099) is used.

servername
The treatment of the *servername* parameter depends on implementation. For the optimized local protocol, this field is ignored. For the `rmi` protocol, it represents the name of the RMI remote object on the remote host.

EXAMPLES

This section presents some examples of monitoring a local JVM with an *lvmid* of 21891.

THE GCUTIL OPTION

This example attaches to lvmid 21891 and takes 7 samples at 250 millisecond intervals and displays the output as specified by the `-gcutil` option.

The output of this example shows that a young generation collection occurred between the third and fourth sample. The collection took 0.078 seconds and promoted objects from the eden space (E) to the old space (O), resulting in an increase of old space utilization from 66.80% to 68.19%. Before the collection, the

survivor space was 97.02% utilized, but after this collection it's 91.03% utilized.

```
jstat -gcutil 21891 250 7
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     G
  0.00  97.02  70.31  66.80  95.52  89.14      7    0.300      0    0.000
  0.00  97.02  86.23  66.80  95.52  89.14      7    0.300      0    0.000
  0.00  97.02  96.53  66.80  95.52  89.14      7    0.300      0    0.000
 91.03   0.00   1.98  68.19  95.89  91.24      8    0.378      0    0.000
 91.03   0.00  15.82  68.19  95.89  91.24      8    0.378      0    0.000
 91.03   0.00  17.80  68.19  95.89  91.24      8    0.378      0    0.000
 91.03   0.00  17.80  68.19  95.89  91.24      8    0.378      0    0.000
```

REPEAT THE COLUMN HEADER STRING

This example attaches to lvmid 21891 and takes samples at 250 millisecond intervals and displays the output as specified by `-gcnew` option. In addition, it uses the `-h3` option to output the column header after every 3 lines of data.

In addition to showing the repeating header string, this example shows that between the second and third samples, a young GC occurred. Its duration was 0.001 seconds. The collection found enough active data that the survivor space 0 utilization (S0U) would have exceeded the desired survivor size (DSS). As a result, objects were promoted to the old generation (not visible in this output), and the tenuring threshold (TT) was lowered from 31 to 2.

Another collection occurs between the fifth and sixth samples. This collection found very few survivors and returned the tenuring threshold to 31.

```
jstat -gcnew -h3 21891 250
 S0C    S1C    S0U    S1U   TT MTT  DSS      EC       EU     YGC     YGCT
  64.0   64.0    0.0   31.7 31  31   32.0    512.0    178.6    249    0.203
  64.0   64.0    0.0   31.7 31  31   32.0    512.0    355.5    249    0.203
  64.0   64.0   35.4    0.0  2  31   32.0    512.0     21.9    250    0.204
 S0C    S1C    S0U    S1U   TT MTT  DSS      EC       EU     YGC     YGCT
  64.0   64.0   35.4    0.0  2  31   32.0    512.0    245.9    250    0.204
  64.0   64.0   35.4    0.0  2  31   32.0    512.0    421.1    250    0.204
  64.0   64.0    0.0   19.0 31  31   32.0    512.0     84.4    251    0.204
 S0C    S1C    S0U    S1U   TT MTT  DSS      EC       EU     YGC     YGCT
  64.0   64.0    0.0   19.0 31  31   32.0    512.0    306.7    251    0.204
```

INCLUDE A TIME STAMP FOR EACH SAMPLE

This example attaches to lvmid 21891 and takes 3 samples at 250 millisecond intervals. The `-t` option is used to generate a time stamp for each sample in the first column.

The Timestamp column reports the elapsed time in seconds since the start of the target JVM. In addition, the `-gcoldcapacity` output shows the old generation capacity (OGC) and the old space capacity (OC) increasing as the heap expands to meet allocation or promotion demands. The old generation capacity (OGC) has grown from 11,696 KB to 13,820 KB after the eighty−first full garbage collection (FGC). The maximum capacity of the generation (and space) is 60,544 KB (OGCMX), so it still has room to expand.

```
Timestamp       OGCMN     OGCMX      OGC        OC        YGC   FGC    FGCT     G
           150.1   1408.0  60544.0  11696.0  11696.0   194    80    2.874
           150.4   1408.0  60544.0  13820.0  13820.0   194    81    2.938
           150.7   1408.0  60544.0  13820.0  13820.0   194    81    2.938
```

MONITOR INSTRUMENTATION FOR A REMOTE JVM

This example attaches to lvmid 40496 on the system named `remote.domain` using the `-gcutil` option, with samples taken every second indefinitely.

The lvmid is combined with the name of the remote host to construct a vmid of `40496@remote.domain`. This vmid results in the use of the `rmi` protocol to communicate to the default `jstatd` server on the remote host. The `jstatd` server is located using the `rmiregistry` command on

`remote.domain` that's bound to the default port of the `rmiregistry` command (port 1099).

```
jstat -gcutil 40496@remote.domain 1000
... output omitted
```